A BOOTSTRAPPING APPROACH TOWARD SHARING OF INFORMATION ON SPACE SYSTEMS

Takahiro Yamada

Institute of Space and Astronautical Science
Sagamihara, Japan
tyamada@pub.isas.ac.jp

ABSTRACT

Many kinds of information is used by engineering teams and application software for spacecraft development and ground systems development, but how the information is structured is varied from team to team and from software to software, even for the same kind of information. The cost of developing spacecraft and ground systems can be reduced by enabling sharing of information among different teams and among different pieces of software. This paper proposes a novel approach toward sharing of information on space systems, in which (1) a generic language for describing structures is defined, (2) domain specific description languages are developed from the generic language, and (3) mapping rules for representing domain specific description languages with standard languages like XML and UML are developed.

1. INTRODUCTION

Many kinds of information is used for spacecraft development and ground systems development. For example, information on how a spacecraft or an instrument is constructed, what commands it receives, and what telemetry it generates is used by many engineering teams and many pieces of software. But how the information is structured is varied from team to team and from software to software, even for the same kind of information. For example, the format of information used by a piece of software is usually determined by the software that uses the information. In other words, information is embedded in the software that uses it (see Figure 1).

However, the cost of developing spacecraft and ground systems can be reduced by enabling sharing of information among different teams and different pieces of software. This becomes possible if we have a method of generating data independently of the team or software that uses it. Data generated in this way can be used by any team or software (see Figure 2).

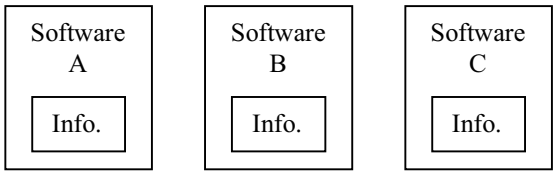

Figure 1: Present Relationship Between Software and Information

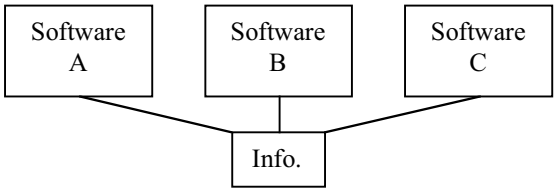

Figure 2: Desirable Relationship Between Software and Information

For example, if the properties of a spacecraft are described with a standard format that does not depend on any engineering team or application software, this information can be shared by many engineering teams and many pieces of application software.

In fact, there are already activities toward sharing of information using standard languages for information description like XML (Extensible Markup Language) [1]. For example, there are several XML-based languages [2]-[4] for describing the command and telemetry data of spacecraft. Each of these languages provides a method of structuring information on command and telemetry data, but these languages are not compatible with each other even though they are very similar to each other, because each language uses a different concept for modelling command and telemetry data.

In the field of software science, there are activities for developing languages for describing software architectures [5], which are collectively known as Architecture Description languages (ADLS). Some space Agencies are

doing researches on how to apply ADLs to space systems. These languages provide methods of structuring information on architectures using a general model of software architectures, but this model is not always applicable to other kinds of architectures (for example, hardware architectures, data architectures, etc.).

This paper proposes a novel approach toward sharing of information on space systems, which is called a bootstrapping approach. The core of this approach is a generic language for describing structures, which can be used to define domain specific models for structuring information in different domains (e.g., spacecraft systems, instruments, software, data units, etc.). From this generic language, various domain specific languages can be developed in a systematic way, and software tools for processing information of multiple domains can be developed.

By using these languages and tools, information on systems can be shared by many engineering teams and pieces of software. This will eliminate the need for re-generating the same information in different formats and re-developing tools to process information in different domains. Therefore, this approach will greatly facilitate reduction of system development cost.

2. BOOTSTRAPPING APPROACH TOWARD SHARING OF INFORMATION

Figure 3 shows the concept of the bootstrapping approach proposed in this paper.

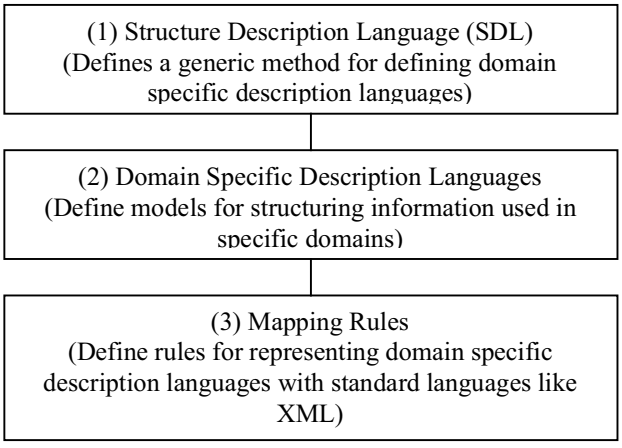

Figure 3. Bootstrapping Approach Toward Sharing of Information

The first step of this approach is to develop a generic language for describing structures. This language, called the Structure Description Language (SDL), provides a mechanism for describing elements of a structure, attributes of elements, and how elements are related with each other. It also has a mechanism for defining specific structural models. By using the capability of SDL to define structural models, SDL can also be used for defining reference models and architectures. In fact, many of the existing reference models and architectures (e.g., Open Systems Interconnection Reference Model (OSI-RM), Reference Model of Open Distributed Systems (RM-ODP), etc.) can be defined (or re-defined) with SDL. Therefore, SDL can also be used as an architecture definition language.

SDL defines five concepts: elements, attributes, classes, relationships, and types. These concepts are explained in section 3.

In each problem domain (such as space systems, distributed systems, communications systems, software, data, etc.), a domain specific description language is developed using the concepts defined in SDL. Domain specific description languages are extensions of SDL and generated by adding domain specific classes and domain specific relationships to SDL. A simple example of domain specific description languages for describing data units is shown in section 4. An example of how an existing Architecture Description Language (ADL) can be re-defined with SDL is shown in section 5.

Both SDL and domain specific description languages are special languages, each having special construction rules and a special vocabulary. But these languages can be processed with existing software tools by defining rules of representing these languages with standard languages like XML. If XML is used, the mapping rule for each domain specific description language can be described as an XML Schema.. By using the Schema, the domain specific language can be processed by appropriate XML tools that are commercially available. Since the process of generating XML Schemas for domain specific description languages is straightforward to experts on XML, examples of XML Schemas are not presented in this paper.

3. STRUCTURE DESCRIPTION LANGUAGE (SDL)

The Structure Description Language (SDL) is a generic language for describing structures. It describes elements of a structure (a system, an architecture, information, a data unit, etc.), attributes of elements, and how elements are related with each other. But it is mostly used for developing domain specific description languages. Therefore, SDL is a meta-language.

SDL does not have any assumption on specific structural models. Domain specific description languages are

developed by defining domain specific structural models using the concepts provided by SDL.

The concepts defined in this language are elements, attributes, classes, relationships and types, which will be explained in the following subsections (see also Figure 4).

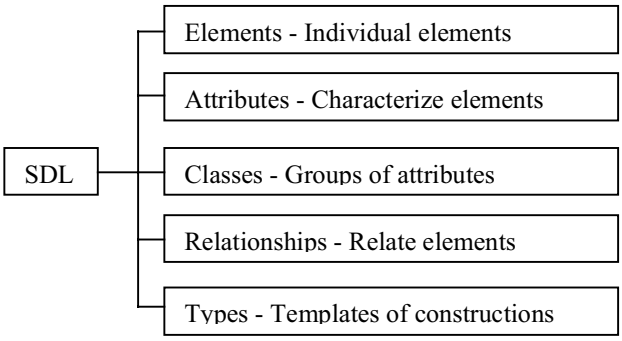

Figure 4. Concepts defined in the Structure Description Language (SDL)

Elements

Elements are things that constitute something. For example, onboard subsystems are elements of a spacecraft. Sections are elements of a document. Spacecraft and documents may be elements of some other things.

Elements may be related to other elements in some way and the relationships between elements are described with the concept of relationships defined in SDL, which will be explained later. An element may have sub-elements and a sub-element may have sub-sub-elements. This kind of containment relationship is also described with the concept of relationships in SDL.

Elements are defined by the users of SDL and domain specific description languages when they define specific structures.

Attributes

Each element has attributes. For example, a spacecraft called `spacecraft-A`, which is an element, has attributes such as `launch date`, `mission-objectives`, `current status`, etc. Each element has a pre-defined set of attributes, and what attributes an element should have is determined by the class (explained below) to which the element belongs.

Each element has a value for each attribute. For example, element `spacecraft-A` has "9 May 2003" as the value of attribute `launch date`. The attributes that an element should have are determined by the class to which the element belongs, but the values of the attributes are specified for each individual element.

Classes

A class is a set of elements that have the same set of attributes. Any element must belong to some class. `spacecraft-A`, which was shown above as an example of an element, belongs to a class called `spacecraft`. Elements are called element instances when, depending on the context, it is important to distinguish individual elements belonging to a class from the class.

The definition of a class includes the definition of the attributes that the elements belonging to that class must have. But the values of the attributes are determined for each individual element as explained above.

Every element must have an attribute called `element-ID`, which is used to identify the element. Therefore, every class must have `element-ID` in its attribute set.

Classes can be related with other classes by the inheritance relationship. For example, the class `planetary-probe` is a sub-class of the class `spacecraft`. Sub-classes inherit the attributes of its super-class; that is, a sub-class must have the attributes of its super-class and usually has additional classes that are specific to that sub-class. For example, the class `planetary-probe` has all the attributes of its super-class `spacecraft`, but it has other attributes like `target body`, `arrival date`, etc.

A sub-class may have sub-sub-classes and an inheritance hierarchy relationship can be established among a set of classes.

Classes are defined by the users of SDL, and the classes that are commonly used in a specific domain are defined in the domain specific description language for that domain.

Relationships

The concept of relationships is used to make assertions about relationships that hold between elements. There are two basic relationships defined in SDL: the containment relationship and the linkage relationship.

The containment relationship is used to express that an element contains another element. SDL has a reserved word for the containment relationship, which is `HAS`. If element A has element B, it is expressed in SDL as "`element-A HAS element-B`."

The linkage relationship is used to express that an element is associated with another element in some way. SDL has a reserved word for the linkage relationship, which is `IS-`

LINKED-TO. If element A is associated with element B in some way, it is expressed in SDL as "element-A IS-LINKED-TO element-B."

If system A has sub-systems B, C and D and they are linked as shown in Figure 5, the relationships between these elements are expressed in SDL as:

```
system-A HAS sub-system-B
system-A HAS sub-system-C
system-A HAS sub-system-D
sub-system-B IS-LINKED-TO sub-system-C
sub-system-B IS-LINKED-TO sub-system-D
```

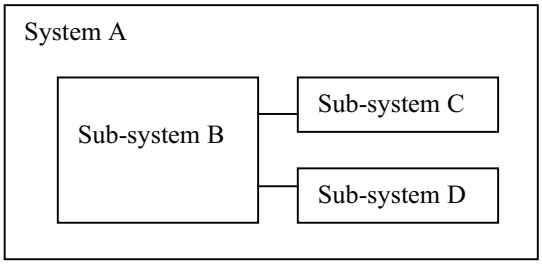

Figure 5. Example of Relationships between Elements

In the above example, System A belongs to some class. Sub-systems B, C and D may belong to a single class, which may or may not be the same as the class of System A, or different classes.

The users of SDL can define variations of the linkage relationship. More complex relationships can also be defined as combinations of the basic relationships. In domain specific description languages, domain specific relationships are defined. An example of a domain specific relationship will be given in Section 4.

Types

Types in SDL is a relatively complex concept. A type is a template for a group of elements that share the same internal structure. Elements belonging to the same type must belong to the same class, but not vice versa. Therefore, types are categories of elements belonging to the same class. But sub-classes are different from types. Sub-classes are defined in terms of how elements are characterizes as a whole, while types are defined in terms of internal structures of elements.

A type specifies:

1) What elements of what classes must be contained in an element of that type;

2) What relationships the contained elements must have; and

3) Some attribute values that the element of that type and its contained elements must have.

Attribute values that are not specified in the type definition are determined for each element.

Types are defined by the users of SDL and domain specific description languages. An example of type definition will be given in Section 4.

Domain Specific Description Languages

Domain specific description languages are derived from SDL by defining domain specific classes and domain specific relationships. Two examples of domain specific description languages are shown in Sections 4 and 5.

4. EXAMPLE OF DOMAIN SPECIFIC DESCRIPTION LANGUAGES (1): A LANGUAGE FOR DESCRIBING DATA UNITS

In this section, a simple language that describes the format of data units is shown. This language defines two domain specific classes and one domain specific relationship.

This language assumes that a data unit consists of either:

1) multiple data fields,
2) multiple data units, or
3) a combination of data fields and data units.

The basic difference between data units and data fields is that a data field has a value but a data unit does not have a value because it has a compound structure. (The word value in this paragraph denotes the value of a data field, not the value of an attribute discussed in Section 3.)

This language defines two domain specific classes: data-unit and data-field. Each of these classes has a set of attributes, as shown in Table 1.

Table 1. Attributes for Describing Data Units

Class	Attributes
data-unit	element-ID explanation length
data-field	element-ID explanation length value encoding

This language also defines a domain specific relationship called IS-SEQUENCE-OF, which is defined as follows:

```
  A IS-SEQUENCE-OF B, C,..., Z
if
  A HAS B,
  A HAS C,
  ...,
  A HAS Z,
  B IS-LINKED-TO C,
  C IS-LINKED-TO-D,
  ...,
  Y IS-LINKED-TO Z
```

In the above definition, A is of type data-unit and B through Z are either of type data-unit or of type data-field. Further, the assertion "A IS-LINKED-TO B" is used to mean that A is followed by B.

The users of this language can define various packets, messages, reports, directives as elements belonging to class data-unit. The users may also define sub-classes of class data-unit and class data-field by defining additional attributes.

For example, let's define an element of class data-unit which is called packet-A as follows:

```
Element: data-unit
 (element-ID = packet-A,
  explanation = "data units for variable-length data
     units",
  length = varied)
```

Let's further suppose that packet-A has two data fields: header and application-data, each of which belongs to class data-field. This is expressed as:

```
Packet-A IS-SEQUENCE-OF
  header: data-field,
  application-data: data-field
```

The users can also define a type, which is a special template for a group of elements. For example, a type called CCSDS-packet can be defied as follows:

```
CCSDS-packet: data-unit
 (element-ID = unspecified,
  explanation = "CCSDS Space Packet",
  length = varied)

CCSDS-packet IS-SEQUENCE-OF
 packet-header: data-unit
  (element-ID = unspecified,
   explanation = "CCSDS Packet Header",
   length = 6 octets),
 application-data: data-unit
  (element-ID = unspecified,
   explanation = "Application data field",
   length = varied)

packet-header IS-SEQUENCE-OF
 version-number: data-field
  (element-ID = version-number1,
   explanation = "Packet version",
   length = 3 bits,
   value = 0,
   encoding = integer),
 packet-type: data-field
  (element-ID = unspecified,
   explanation = "Packet type",
   length = 1 bit,
   value = unspecified),
   encoding = integer),
 ...
```

The above type definition specifies that:

1) Elements of type CCSDS-packet must have packet-header and application-data, both of which are data units.

2) packet-header must have version-number, packet-type, and so on, which are data fields.

3) Some attributes must have specific values. For example, the value of attribute length of version-number of packet-header must be 3 for any element of this type. The attribute values shown as "unspecified" in the type definition are determined for each element.

There are already languages for describing data units [2]-[4], but this approach can be used to enhance the usability of existing languages in the following way. Existing languages can be re-defined as domain specific description languages with the concepts of SDL. Using this re-definition of existing languages, translation rules between different languages can be generated so that information generated with a tool based on one language can be processed with another tool based on another language. Therefore, this approach can be used for making different languages and different tools interoperable with each other.

5. EXAMPLE OF DOMAIN SPECIFIC DESCRIPTION LANGUAGES (2): AN ARCHITECTURE DESCRIPTION LANGUAGE

There are languages for describing software architectures [5], which are collectively known as Architecture Definition Languages (ADLs). Some of theses ADLs can be re-defined as domain specific description languages

with SDL by re-defining the basic components of the original languages as domain specific classes and the construction rules of the original languages as domain specific relationships.

Such an example for an architecture definition language called xADL [6] is shown below. This language describes software architectures using the following concepts:

1) Components (loci of computation),
2) Connectors (loci of communication),
3) Interfaces (exposed entry and end points), and
4) Links (concept to define topological arrangements).

These concepts can be re-defined with the concepts of SDL as follows:

1) Components => a domain specific class,
2) Connectors => a domain specific class,
3) Interfaces => a domain specific class, and
4) Links => a domain specific linkage relationship.

xADL also has a mechanism for defining sub-architectures and this can be re-defined with the containment relationship of SDL.

SDL still lacks the capability of describing dynamic behavior of elements, which we hope to add later. But portions of ADLs that describe static characteristics of software architectures can be re-defined with SDL. Therefore, partial translation between ADLs will be possible using SDL as a language definition language.

6. FUTURE WORK ITEMS

The work presented in this paper is still at a preliminary stage, and there are many items that have to be worked out to complete this approach. The following is a list of such open items.

1) Semantic information. We need a unified method for describnibing semantic information associated with elements.

2) Dynamic behavior. We need a method for describing dynamic behavior of elements.

3) Representation with graphical modeling languages. We need to consider how SDL and domain specific description languages are represented with graphical modeling languages like Unified Modeling Language (UML).

7. CONCLUSION

In this paper, we have shown a novel approach toward sharing of information on space systems and other systems. This approach consists of three steps: (1) a generic language for describing structures is defined, (2) domain specific description languages are developed from the generic language, and (3) mapping rules for representing domain specific description languages with standard languages like XML are developed.

We have presented the concepts of the Structure Description Language (SDL) and shown how its concepts are used to develop domain specific languages using two examples. Examples of step 3 were not shown here because step 3 is straightforward, if XML is used, by using the capability of XML Schema.

By using this approach, various kinds of information can be shared by many engineering teams and pieces of software. This approach also enables generation of software tools that do not depend on any specific domain. Also, tools developed for specific domains can be used in other domains, and tools developed for specific languages can be used for processing data described with other languages. The cost of developing spacecraft and ground systems will be greatly reduced by this approach.

8. REFERENCES

[1] W3C, “Extensible Markup Language (XML) 1.0,” REC-xml-19980210, W3C Recommendation, February 1998.

[2] T. A. Ames, K. B. Sall, C. E. Warsaw, and R. A. Shafer, “Using XML for Instrument Description, Communication and Control of the SOFIA/HAWC Instrument,” *AAS Meeting #193*, Austin, TX, January 1999.

[3] S. Jann and , R. Gresser, “The FUSE Database: A Single Source Database used throughout the Life Cycle of the Mission: Bench Test Environment, Integration and Test, and Operations,” *SpaceOps 2000*, Toulouse, France, June 2000.

[4] T. Burch, “Using a Generic XML Driven Translator for Downlink and Uplink Data,” *First Joint Space Internet Workshop*, Greenbelt, MD, November 2000.

[5] N. Medidovic, R. N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages," *IEEE Transactions on Software Engineering*, Vol. 26, No. 1, pp.70-93, January 2000.

[6] R. Khare, et al., "xADL: Enabling Architecture-Centric Tool Integration with XML," *Proceedings of the 34th Annual Hawaii International Conference on System Science (HICSS-34)*, 2001.